Asynchronous Covert Communication Using BitTorrent Trackers

Mathieu Cunche, Mohamed Ali Kaafar, Roksana Boreli

Asynchronous Covert Communication Using BitTorrent Trackers

Mathieu Cunche*†, Mohamed-Ali Kaafar†‡, Roksana Boreli‡
†Inria, France *INSA-Lyon CITI, France ‡National ICT Australia
firstname.lastname@inria.fr *firstname.lastname*@nicta.com.au

***Abstract*—Covert channels enable communicating parties to exchange messages without being detected by an external observer. We propose a novel covert channel mechanism based on BitTorrent trackers. The proposed mechanism uses common HTTP commands, thus having the appearance of genuine web traffic and consists of communications that are both indirect and asynchronous: no messages are directly exchanged between the sender and the receiver (of covert communications) and there is a potentially considerable delay between the sender's message to the relaying party and the receiver collecting this message. We present details of the proposed scheme in which a centralized BitTorrent tracker is used for storing covert messages and evaluate its performance based on the implemented prototype. We analyze the detectability of covert communications by an adversary and show that, while the common nature of the BitTorrent traffic and the large number of clients make the detection unlikely, the low temporal correlation between the writer and the reader (the two communicating parties) further increases the detection difficulty.**

I. Introduction

While the purpose of encryption is to protect the content of any communication from being revealed to unauthorized parties, covert channels aim to hide the existence of the communication itself. Covert channels were first described in the context of Multi-Level Security (MLS) systems [3], as a way to communicate information from high security to low security processes. They have been subsequently extended to the case of network communications, with their application ranging from military and national security, where they are used to hide potential communications between parties, to botnets and use by cyber criminals, that look to control remote systems while circumventing traffic filtering.

The huge amount of data exchanged through Peer-to-Peer (P2P) networks and the global-scale of P2P infrastructure creates an environment where communication has a potential to be hidden among the mass of regular users and corresponding host message exchanges. These feature has been exploited to create covert communications channels [7], [4]. However those system, like most covert channels [5], rely on a direct connection between the communicating parties.

Contrarily to those works, we propose to use the P2P infrastructure, to build a novel asynchronous (network storage) covert channel that does not require the direct communications. The key insight behind our approach is to leverage the services provided by the BitTorrent tracking infrastructure, a key element that maintains an inventory of the peers involved in a swarm (set of peers downloading and/or sharing a given content). Our contributions are as follows.

We present a communication scheme that enables two parties to perform a hidden exchange of information through the centralized BitTorrent tracker. Two inherent properties of the scheme are beneficial to covert communications, as they assist with having a low detectability: the scheme is an indirect network covert channel and it is asynchronous, with the sending and retrieving nodes having the flexibility of not needing to access the intermediate node at the same time. The scheme includes error correction and encryption mechanisms, respectively, to mitigate data losses and to provide obfuscation of transmitted data. Our system exploits the basic primitives of the (centralized) BitTorrent tracker protocol, and therefore does not require the cooperation of the tracker.

Based on a prototype implementation we show that our system can reliably transmit data at an average rate of 1.89 bits/second, and that the data remain available for dozens of minutes, providing a considerable time window in which the receiver may retrieve the data. Overall these results demonstrate that such a communication channel can reliably transmit information at a rate sufficient for the exchange of small pieces of data (such as text messages or botnet commands) without a fine-grained time synchronization between the sender and the receiver.

We analyze the detectability of our scheme, which includes a number of features well suited to covert communications. First, our system is solely based on one of the most commonly used operation on the Internet, i.e. the HTTP/GET request, resulting in communications having the appearance of genuine traffic. Second, it is asymmetric in the sense that read and write operations are not equivalent, resulting in a distinct cost and a detection difficulty for the sender and the receiver.

II. Centralized BitTorrent Trackers

BitTorrent is a Peer-to-Peer system in which peers share and download content by exchanging content chunks. BitTorrent rely on a centralized server called *tracker*, to maintain and distribute the list of the peers involved in a swarm (set of peers downloading and/or sharing a given content). Each centralized tracker usually stores the list of peers for several contents. For instance, we found several millions of contents registered on the *publicbt* tracker.

The BitTorrent system uses a content identification system based on hash called *infohash* obtained by computing the

SHA-1 digest of the content meta-information. Therefore an infohash is a 20 bytes long (160 bits) identifier. In order to obtain the list of peers connected to a swarm, a BitTorrent client contacts a centralized tracker using an *Announce* request. This request can be performed through either UDP or HTTP protocol. In both cases the requested URL contains several parameters, in particular the content's identifier: http://[tracker_url]/announce?info_hash=x.

Upon reception of an announce request, and if the content is already registered on the tracker, the tracker returns a set of peers and adds the requesting peer to the list. If the content is not registered on the tracker, there are two possible cases depending on the tracker policy: either the tracker rejects the request by specifying a request for an unregistered content, or the tracker registers the content and adds the requesting peer to the corresponding peer list. In the later case the tracker is called *open*, in the sense that any peer can register a new content on the tracker by performing an announce request. Within the BitTorrent ecosystem, both open and non-open centralized trackers co-exist.

Additionally, centralized BitTorrent trackers support other requests such as the *scrape-all* request that provides the full list of the contents registered on the tracker. A *scrape-all* request is performed through an HTTP/GET query on the following url : http://[tracker_url]/scrape . In order to remove the burden of the *scrape-all* operation from the tracker itself, the corresponding information is sometime offered by a web server. For instance, the *publicbt* tracker provides a compressed scrape file refreshed every 10 minutes at the following address : http://publicbt.com/all.txt.bz2.

III. Centralized Tracker-Based Covert Channel Design

In this section, we outline the design of the proposed asynchronous cover channel.

Read and Write operations: The centralized tracker can be used as a memory in which the information is stored under the form of data block represented by *info_hash* identifier. Since the *info_hash* used as content identifier is 20 bytes long, the information can be stored on the tracker as chunk of 20 bytes (160 bits).

Writing on the tracker can be done by adding an entry using an announce request. Let x be a 20 bytes length value, then sending the following request to the tracker will add an entry x to the content table: `GET http://[tracker_url]/announce?info_hash=x`.

To read a value from the tracker, we have to get the list of the content identifier stored on the tracker. This operations can be done by using the *scrape all* request on the tracker. As noted in Section II, some tracker, such as publicbt.com, provide this information under the form of an aggregated file (we used the compressed text file provided by publicbt.com that can be retrieved at http://publicbt.com/all.txt.bz2).

Hash-based filtering for packet identifications: The previously described read operation retrieve all the *info_hash* stored in the content table. This content table contains both *genuine* entries, i.e. entries corresponding to existing torrent, and *hidden* entries, i.e. entries added to the tracker by our hidden communications. To distinguish *hidden* entries from *genuine* ones, we use a hash-based filtering mechanism. For each piece of information stored on the tracker, the last t bits are a hash value of the $160 - t$ first bits, using a hash function, $h(.)$. For the read operation, after the *scrape all* operation has been performed, the retrieved *info_hash* are filtered. Specifically, for each *info_hash* x, keep it only if $h(x[0 \ \ldots \ 160 - t - 1]) = x[160 - t \ \ldots \ 160 - 1]$.

This hash-based filtering approach may however induce false positives, where *info_hash* may pass the filter while they are genuine and not hidden entries. We have computed [2] that a 4-bytes long hash ($t = 32$) yields to $6.67\ 10^{-5}$ false positives on the average, which we consider as a good compromise between communication overhead induced by the hash size and the expected number of transmission failure caused by false positives.

Obfuscation through encryption: With the previous hash-based filtering mechanism, the *hidden info_hash* can be easily detected by someone knowing the algorithm. This issue can be avoided by encrypting the data before it is written on the tracker, for instance with a symmetric block-cipher. Indeed, only a person knowing the secret key will be able to decrypt the retrieved *info_hash* and filter out those that are corresponding to hidden pieces of information. In addition, this encryption provide data confidentiality and in particular prevent an eavesdropper to recognize the hidden data in plain text. Note, that the key can be shared in advance, for instance by embeding it in the core of a botnet node.

Data indexing: The size of the *info_hash* limits the amount of data that can be transmitted in one entry of the content table, which is even more reduced with the adjunction of the hash of the useful data. In essence, a total of $160-t$ bits of information can transmitted per entry. In the case where a larger amount of data has to be transmitted, the data can be split and carried by several content identifiers. In order to transmit a message split in several pieces, we need to add an indexing functionality. To this aim we reserve the first s bits of data for indexing. Those s bits simply contain a sequence identifier, and are concatenated to the data before the addition of the hash to allow the indexing of up to 2^s pieces of information.

Data loss and erasure coding: During our experiment, we have observed that the data written on the tracker can be subject to erasure[1], i.e. loss of information.

We propose to solve this issue of data losses by using a Reed-Solomon coding scheme [6]. After splitting the message into k blocks of $160 - t - s$ bits, the encoding process is applied and n blocks, also of $160-t-s$ bits, are generated as output of the encoder. Thanks to the optimal erasure correction capabilities of the Reed-Solomon codes, the reader can recover the source data as soon as k blocks have been collected. The

[1]Data written on the tracker was not returned in the *scrape-all* results.

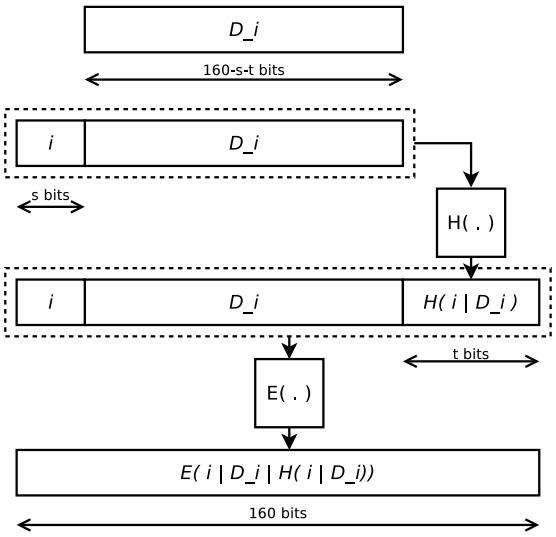

Fig. 1. Creation of the forged content identifier.

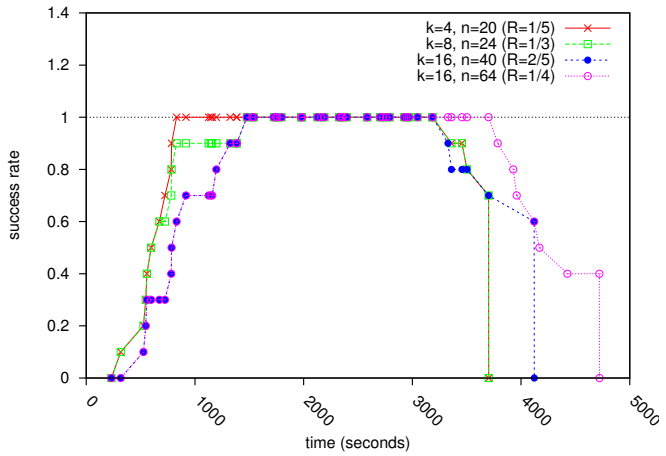

Fig. 2. Success rate as a function of the time elapsed since the end of the first write operation.

amount of redundancy represented by the code rate, $R = n/k$, can be adjusted to meet the channel requirements, i.e. as appropriate to the loss rate. Experimental results [2] have shown a loss rate of 44.92%, that can be compensated by using a coding rate $R < 1/2$.

Summary of the covert channel mechanism: On the writer side, the message is first divided into k blocks of $160 - t - s$ bits and encoded into n blocks. For each block, an index and a hash are then added and the obtained data is encrypted before being written (through a HTTP Get Request) on the tracker. Figure 1 describes the encoding of a single data block. To retrieve the message, the reader performs a *scrape-all* request, then decrypts all the content identifiers and uses the hash-based filter to select the hidden data blocks. If enough blocks have been collected, a decoding process is applied to decode the original message.

IV. Performance Analysis

This section studies the performance of the proposed communication system in term of transmission success rate and the throughput. We consider a scenario were a unique sender wants to transmit a message to a unique receiver using the following configuration : hash-size $t = 32$ bits, index size $s = 6$ bits, which leave $160 - 6 - 32 = 122$ bits of payload in *info_hash*[2]. A message is divided into k blocks of 122 bits, that are then encoded by the erasure protection scheme into n block of 122 bits.

Transmission success rate: As explained in Section III, the potential data losses can be recovered by using an error correcting code [6]. Several configuration with various message sizes $(2, 4, 8, 16$ data blocks) and code rates ($1/5$ and $1/2$) have been tested. During this experiments, write operations are performed in parallel by 5 threads while read operations are performed at regular interval (every 10 minutes). We then compute the success rate of a read operation as a function of the time elapsed since the end of the first write operation. A read operation is considered successful if the information contained in the retrieved scrape file is sufficient to decode the message.

Fig. 2 presents the result corresponding to the configurations that have provided the best performances. Focusing on the the selected configurations, there is a time frame during which a read operation is always successful. The length of this time frame depends on the parameters, and ranges as depicted on Fig. 2, from 1700 seconds to 2350 seconds, allowing for asynchronous transmission as the message can be read anytime during a time frame of several tenths of minutes.

Transmission throughput: The proposed hidden channel will have an associated throughput, i.e. the data rate at which the sender will be able to transmit data to the receiver.

We note that to transmit a message, a number of write operations need to be done (one for each block i.e. elementary message), while only one elementary read operation is required (the *scrape-all* operation). Consequently, the time to write a message will depend both on the message size (number of blocks) and the elementary write delay, while the read operation will result in a constant delay. Assuming a large number of write operations, the throughput can therefore be approximated by the throughput of the write operation.

Table IV presents a summary of the throughput results, for the following parameters: $s = 6$ and $t = 32$, with 122 of useful bits per data block that can be used to transmit information. With the elementary read operation resulting in an average delay of 111.289 seconds and the elementary write operation an average delay of 65.475 seconds, the resulting write throughput (and the system throughput) is 1.89 bits per seconds. Note that performing the write operations in parallel has a potential to increase the system throughput.

V. Detectability Analysis

We now discuss the difficulty of detecting our covert-channel based communication. We consider two types of

[2]We used the open tracker publicbt.com

TABLE I
SUMMARY OF THE DELAY INTRODUCED BY THE ELEMENTARY OPERATIONS AND THE RESULTING THROUGHPUT.

Operation	Elementary operation time min / avg / max	Average throughput
Write	0 / 65.475 / 532 s	1.89 b/s
Read	86 / 111.289 / 175 s	NA

wardens: a weak adversary (monitoring warden) that has no knowledge of the scheme used for communications, and a strong adversary that knows all the parameters of the used scheme, with the exception of the secret key used to encrypt the data, as discussed in section III.

The goal of a warden ignoring the proposed protocol is to spot communication corresponding to our scheme. We will now discuss the characteristics of the traffic generated by our scheme, and how they can be used to be classified as *suspicious* by the warden.

Because our scheme solely rely on HTTP GET requests, which are one of the most common message used on the Internet, the traffic generated by the communicating parties will look like genuine web traffic to the warden. The warden could use the requested content to identify the targeted application, i.e. BitTorrent. However while BitTorrent system, may not be as common as web services, it is still one of the most popular services of the Internet and has been reportedly used by millions of users[3].

The goal of a warden aware of the protocol could be to detect the communicating parties and the traffic corresponding to our scheme. Also, the warden may already know the identity of one party and could try to identify the other party.

The warden should first focus on the traffic directed to the centralized BitTorrent trackers, and more particularly to trackers providing the functionality required by our scheme (open centralized tracker supporting scrape-all requests). All the BitTorrent clients performing *announce* requests should be considered as potential writers while all the clients performing *scrape-all* requests should be considered as readers. As stated before, the second category can be rather small while the first can be very large. Indeed, the number of clients connected to a tracker at the same time can scale up to 100.000's peers [1]. To the best of our knowledge, no information on the number of clients performing scrape-all requests is available. Nevertheless, given the specific nature of this request, we can conjecture that the number of clients will be small. The latter property makes our system asymmetric in terms of detection difficulty: discovering the identity of the writer is much more difficult than discovering that of the reader. This feature suits well the scenario where an insider is attempting to exfiltrate information to the outside.

Announce requests performed by the writer are using content identifier forged for the purpose of the protocol, and are therefore not corresponding to real content shared on the BitTorrent ecosystem. List of contents shared in the BitTorrent ecosystem can be found on websites like *The Pirate Bay*. The warden could use this information to verify that the identifier corresponds to real content, but in fact there is a significant part of content shared in the BitTorrent ecosystem that is not found on the common *torrent* databases. Therefore, this technique could be used to narrow down the search, but will not be enough to identify the writer.

Finally we want to highlight two features of our scheme that make the detection even more difficult. First, as opposed to many other covert channels, no direct connection is ever established between the communicating parties. This means that the warden may not even suspect that the parties are exchanging information. Second the system is asynchronous, i.e. the read operation can be done several ten's of minutes after the write operation. Thanks to this feature the temporal correlation between the writer and the reader activity can be kept at a low level.

VI. CONCLUSION

We have presented a new covert channel based on a widely used service available on the Internet: centralized BitTorrent trackers. Our system does not require the cooperation of the tracker and allows the sending of data at a moderate rate. We have implemented a prototype that demonstrates the practicality of our proposal and evaluated the performance of our system. By analyzing the scheme under two attacker models, with the monitoring warden being either the strong or the weak adversary, we have shown that the nature of the communications involved in the proposed scheme makes the detection and blocking of the communications difficult. Finally, this scheme could extended to a decentralized, DHT based, trackers in order to improve scalability and further reduce detectability [2].

REFERENCES

[1] Nazareno Andrade, Miranda Mowbray, Aliandro Lima, Gustavo Wagner, and Matei Ripeanu. Influences on cooperation in bittorrent communities. In *Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, P2PECON '05, pages 111–115, New York, NY, USA, 2005. ACM.

[2] Mathieu Cunche, Mohamed Ali Kaafar, and Roksana Boreli. Asynchronous Covert Communication Using BitTorrent Trackers. Rapport de recherche RR-8554, INRIA, June 2014.

[3] Department of Defense. *Department of Defense Trusted Computer System Evaluation Criteria*, December 1985. DOD 5200.28-STD (supersedes CSC-STD-001-83).

[4] R. Eidenbenz, T. Locher, and R. Wattenhofer. Hidden communication in P2P networks steganographic handshake and broadcast. In *INFOCOM, 2011 Proceedings IEEE*, pages 954 –962, april 2011.

[5] Norka Lucena, Grzegorz Lewandowski, and Steve Chapin. Covert channels in ipv6. In *Privacy Enhancing Technologies*, volume 3856 of *Lecture Notes in Computer Science*, pages 147–166. Springer Berlin / Heidelberg, 2006.

[6] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 27(2), April 1997.

[7] Guenther Starnberger, Christopher Kruegel, and Engin Kirda. Overbot - a botnet protocol based on kademlia. In *4th International Conference on Security and Privacy in Communication Networks*, 2008.

[3] http://torrentfreak.com/bittorrent-surges-to-150-million-monthly-users-120109/

.